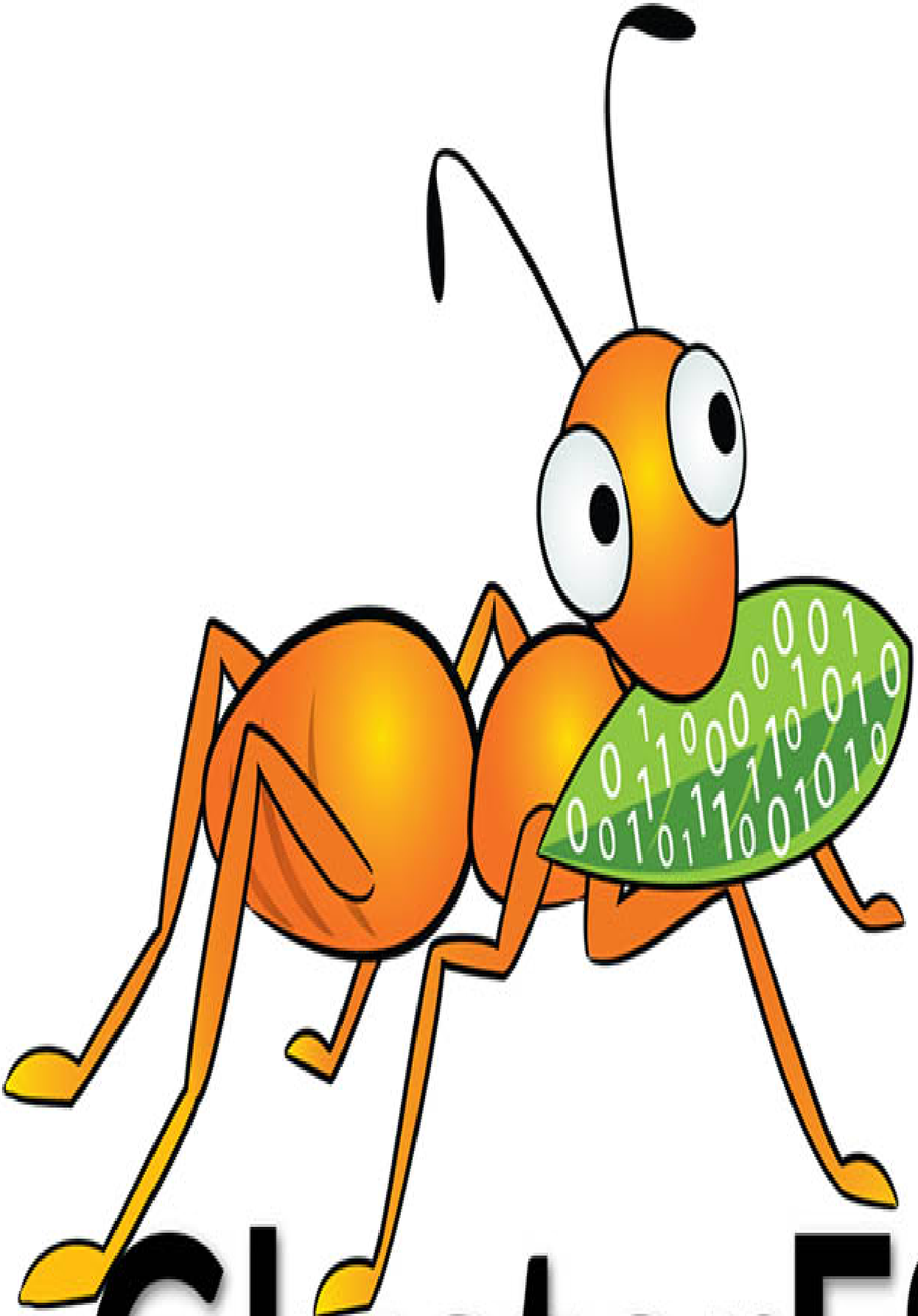

GlusterFS

GlusterFS

In this post we focus on GlusterFS,
which is a distributed file system developed by RedHat.
According to RedHat,

> GlusterFS is a scalable network filesystem. Using common off-the-shelf hardware, you can create large, distributed storage solutions for media streaming, data analysis, and other data- and bandwidth-intensive tasks. GlusterFS is free and open source software.

We will see if we can install GlusterFS on the Raspberry Pis in order to create a fast and highly available filesystem.

Networking setup

Before we start installing GlusterFS we need to make sure the nodes can find each other on the network.
To achieve this,
we add the hostnames of the other nodes to each nodes' hosts file, found at /etc/hosts. For example, to the hosts file of the first node we add the following three lines:

```
10.0.1.1  rpi1
10.0.1.2  rpi2
10.0.1.3  rpi3
```

We can test the new configuration by pinging the other nodes:

```
user@rpi0~$ ping rpi1 -c 1
PING rpi1 (10.0.1.1) 56(84) bytes of data.
64 bytes from rpi1 (10.0.1.1): icmp_seq=1 ttl=64 time=0.608 ms
```

Installing GlusterFS

The installation of the GlusterFS package is very simple: running

```
user@rpi0~$ sudo apt-get install glusterfs-server
```

will install all required packages on a node. Repeat this on each node.

Probing the peers

Now we need to introduce the individual GlusterFS-servers to each other. This is done with the peer probe command:

```
user@rpi0~$ sudo gluster peer probe rpi1
peer probe: success.
user@rpi0~$ sudo gluster peer probe rpi2
peer probe: success.
user@rpi0~$ sudo gluster peer probe rpi3
peer probe: success.
```

After probing the peers they should show up in the peer status list:

```
user@rpi0~$ sudo gluster peer status
Number of Peers: 3

Hostname: rpi1
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)

Hostname: rpi2
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)

Hostname: rpi3
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)
```

Probing a peer on one of the nodes automatically adds the node to the peer lists of all nodes. If we have a look at the peer status list on one of the other nodes the peers show up as well:

```
user@rpi1~$ sudo gluster peer status
Number of Peers: 3

Hostname: 10.0.1.0
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)

Hostname: 10.0.1.2
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)

Hostname: 10.0.1.3
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)
```

Note that the ip addresses of the other nodes show up in the list, instead of the hostnames.
This can be fixed by probing the peers in reverse. After running

```
user@rpi1~$ sudo gluster peer probe rpi0
peer probe: success.
user@rpi1~$ sudo gluster peer probe rpi2
peer probe: success.
user@rpi1~$ sudo gluster peer probe rpi3
peer probe: success.
```

the hostnames should be visible in the peer status list:

```
user@rpi1~$ sudo gluster peer status
Number of Peers: 3

Hostname: rpi0
```

```
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)

Hostname: rpi2
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)

Hostname: rpi3
Uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
State: Peer in Cluster (Connected)
```

You may need to repeat this process on the remaining nodes as well.

Creating a volume

Now we're ready to create a volume. A volume is a collection of *bricks*, where a brick is a directory on one of the servers. The most basic kind of volume is the **distributed** volume, in which the files are distributed randomly over the bricks in the volume. A distributed volume therefore has no redundancy, which makes it very susceptible to failure.
We can create a distributed volume with two bricks with the following command:

```
gluster volume create  transport tcp server1:/exp1 server2:/exp2
```

In a **replicated** volume every file is replicated in multiple bricks, providing redundancy.
The number of bricks in
a replicated volume should be a multiple of the number of replicas required. If the number of bricks is larger than
the number of replicas, the volume is called a **replicated distributed** volume.
We can create a replicated volume with two bricks and a replica count of 2 with the following command:

```
gluster volume create  replica 2 transport tcp server1:/exp1 server2:/exp2
```

If you are not interested in redundancy but in reading very large files fast you should consider a **striped** volume, in which files are striped over multiple bricks. Like with the replicated volumes the number of bricks in a striped volume should be a multiple of the stripe count, and if the number of bricks is larger than the stripe count it becomes a **striped distributed** volume. Creating a striped volume with two bricks and a stripe count of 2 is done with the following command:

```
gluster volume create  stripe 2 transport tcp server1:/exp1 server2:/exp2
```

If you have multiple disks in each server you can create a volume with more than one brick per server:

```
gluster volume create  transport tcp server1:/exp1 server1:/exp2 server2:/exp3 server2:/exp4
```

When making a distributed striped or distributed replicated volume with multiple bricks on each server you should keep in mind that the replica and stripe groups are established by the order you specify the volumes. For example, in a volume with replica count 2, and 4 bricks on 2 servers,

```
gluster volume create  replica 2 transport tcp server1:/exp1 server1:/exp2 server2:/exp3 server2:/exp4
```

the first two bricks will form a replica set, as will the last two bricks. In this case, that means that the replicas will live on the same server, which is generally a bad idea. Instead one can order the bricks such that files are replicated on both servers:

```
gluster volume create  replica 2 transport tcp server1:/exp1 server2:/exp3 server1:/exp2 server2:/exp4
```

We will create a replicated distributed volume called gv, with a replica count of 2.
We will place the bricks on USB flash drives, mounted on /mnt/usb on each of the nodes, in folders called gv:

```
user@rpi0~$ sudo gluster volume create gv replica 2 transport tcp
rpi0:/mnt/usb/gv rpi1:/mnt/usb/gv rpi2:/mnt/usb/gv rpi3:/mnt/usb/gv
volume create: gv: success: please start the volume to access data
```

If the creation succeeded, we can start the volume with:

```
user@rpi0~$ sudo gluster volume start gv
volume start: gv: success
```

after which we can check that it is in fact started:

```
user@rpi0~$ sudo gluster volume info

Volume Name: gv
Type: Distributed-Replicate
Volume ID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
Status: Started
Number of Bricks: 2 x 2 = 4
Transport-type: tcp
Bricks:
Brick1: rpi0:/mnt/usb/gv
Brick2: rpi1:/mnt/usb/gv
Brick3: rpi2:/mnt/usb/gv
Brick4: rpi3:/mnt/usb/gv
```

Mounting a volume

Now that the volume has been started, mounting it is a piece of cake. First create a mount point (e.g. /mnt/gluster), and then mount the volume with:

```
user@rpi0~$ sudo mount -t glusterfs rpi0:/gv /mnt/gluster/
```

You can point gluster to any of your nodes that are part of the volume (so rpi1:/gv would work just as well) as long as they are up at the time of mounting: the glusterfs client only uses this node to obtain a file describing the volume. In fact, you can connect the client with *any* node, even one that is not part of the volume, as long as it is in the same peer group as the volume.

Once mounted, the volume will show up in the list of filesystems:

```
user@rpi0~$ df
Filesystem  1K-blocks  Used Available Use% Mounted on
/dev/root  31134100  1459216  28382804  5% /
devtmpfs  218416  0  218416  0% /dev
tmpfs  222688  0  222688  0% /dev/shm
tmpfs  222688  4484  218204  3% /run
tmpfs  5120  4  5116  1% /run/lock
tmpfs  222688  0  222688  0% /sys/fs/cgroup
/dev/mmcblk0p1  61384  20480  40904  34% /boot
/dev/sda1  7658104  19208  7226840  1% /mnt/usb
rpi0:/gv  15316096  35584  14456448  1% /mnt/gluster
```

Note that its size is almost twice that of the usb drive in /mnt/usb, which is what we expect as the volume is distributed over four usb drives, and all data is replicated twice.

If you want the Gluster volume to mount every time you start a server, you can add the mount point to your /etc/fstab file. In our example we would add a line like this:

```
rpi0:/gv /mnt/gluster glusterfs defaults,_netdev 0 0
```

Mounting on other machines

In order to mount the volume on another machine you'll have to install the GlusterFS client. On Debian-based distributions
this can be done by running sudo apt-get install glusterfs-client. Additionally you'll have to add (one of) the glusterfs server(s) to the /etc/hosts file, as described above for the peers.

You may run into problems if the version of GlusterFS that is provided in the glusterfs-client package on your client machine is not the same as the one that is provided in the glusterfs-server package on your servers. In this case you can
instead install the client from source.

Performance

To test the performance of the newly created filesystem we will use a fifth Raspberry Pi, on which I mounted the volume as described above. Keep in mind that since this also a Raspberry Pi all performance measures below could be (are) also limited by the computing power on the client side.

Writing performance

We can test the raw writing speed to the GlusterFS filesystem by using dd to write a 64 megabyte file using a varying block size:

```
user@client~$ dd if=/dev/urandom of=/mnt/gluster/file1.rnd count=65536
bs=1024
65536+0 records in
65536+0 records out
67108864 bytes (67 MB) copied, 239.101 s, 281 kB/s
```

The achieved speed, 281 kB/s, is far from impressive. Increasing the block size to 128 kB increases this speed significantly:

```
user@client~$ dd if=/dev/urandom of=/mnt/gluster/file2.rnd bs=131072
count=512
512+0 records in
512+0 records out
67108864 bytes (67 MB) copied, 67.4289 s, 995 kB/s
```

This speed is not too far from the maximum reading speed of /dev/random:

```
user@client~$ dd if=/dev/urandom of=/dev/null count=65536 bs=1024
65536+0 records in
65536+0 records out
67108864 bytes (67 MB) copied, 42.3208 s, 1.6 MB/s
```

Reading performance

Reading a file is much faster than writing, as you may have expected:

```
user@client~$ dd if=/mnt/gluster/file2.rnd of=/dev/null bs=131072
512+0 records in
512+0 records out
67108864 bytes (67 MB) copied, 9.89423 s, 6.8 MB/s
```

Once read, the file remains cached, as you can see when you read it again:

```
user@client~$ dd if=/mnt/gluster/file2.rnd of=/dev/null bs=131072
512+0 records in
512+0 records out
67108864 bytes (67 MB) copied, 0.252037 s, 266 MB/s
```

Small files

It is when performing operations on many small files that the performance really drops. For example, if we create a folder containing 1024 files, each of 4 kB:

```
user@client~$ mkdir /mnt/gluster/dir1/
```

```
user@client~$ for i in {1..1024}; do dd if=/dev/urandom
of=/mnt/gluster/dir1/file${i}.rnd bs=4096 count=1; done
```

and then recursively copy this folder into another folder:

```
user@client~$ time cp -r /mnt/gluster/dir1 /mnt/gluster/dir2

real  3m15.665s
user  0m0.120s
sys 0m1.900s
```

it takes over three **minutes** to copy some 4 MB! By comparison, on the Raspberry Pi's SD card (which is fairly slow), this takes about half a second:

```
user@client~$ time cp -r ~/dir1 ~/dir2

real  0m0.612s
user  0m0.020s
sys 0m0.590s
```

Availability

One of the main promises of GlusterFS is redundancy. We can easily test this by shutting down one of the servers, after which the filesystem will respond like nothing happened. You can even shut down two of the servers, as long as you do not shut down two servers in the same replica set (in our example, rpi0 and rpi2 or rpi1 and rpi3).

Even if you temporarily shut down two servers that are in the same replica set after each other (e.g. reboot rpi0, wait a few minutes, then reboot rpi2), things will carry on like nothing happened, as GlusterFS automatically distributes the changes when the brick comes back up. However, if you do not give it enough time to resync things go wrong, resulting in errors like:

```
user@client/mnt/gluster$ rm -r *
```

```
rm: cannot remove ‘dir1’: Transport endpoint is not connected
rm: cannot remove ‘dir2’: Transport endpoint is not connected
rm: cannot remove ‘file3.rnd’: Invalid argument
rm: cannot remove ‘file4.rnd’: Invalid argument
```

Some of these errors you may be able to fix by re-mounting the volume on the client or perhaps by restarting all servers. If that does not the problem, try rebalancing the volume. On one of the servers, run:

```
user@rpi0~$ sudo gluster volume rebalance gv fix-layout start
```

Conclusion

Although the Raspberry Pis are clearly not made to run this kind of filesystem (and of course GlusterFS was not made to run on a Raspberry Pi) it is perfectly possible to create your own ~~high performance and~~ highly available filesystem. Even if one of your Raspberry Pis crashes (mine do once in a while) you will still be able to access all your files.